



**High Level Design (OOAD)**

<Project Name>

<Project Id>

Document Version / Details : <ver no. / Date>

**Record of Release**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version No. | Modified By | Reviewed By | Authorized By | Release Date | Modifications Done |
| <Ver.> | <Full Name> | <Full Name> | <Full Name> | <Release Date> |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Table of Contents

[1 Introduction & Brief Description 5](#_Toc484523965)

[2 References 6](#_Toc484523966)

[3 Basic assumptions and limitations of the system 7](#_Toc484523967)

[4 System considerations, Architectural analysis, Design & Development decisions and rationale 8](#_Toc484523968)

[5 Use-Case View 17](#_Toc484523969)

[5.1 Architecturally-Significant Use Case diagrams 17](#_Toc484523970)

[6 Logical View 18](#_Toc484523971)

[6.1 Application Architecture: 18](#_Toc484523972)

[6.2 Architectural Pattern 19](#_Toc484523973)

[6.3 Important layers 20](#_Toc484523974)

[6.4 Architecturally-Significant Model Elements 21](#_Toc484523975)

[6.5 Package and Subsystem Layering 22](#_Toc484523976)

[6.6 Architectural Description 23](#_Toc484523977)

[7 Process View 24](#_Toc484523978)

[7.1 Class diagram 24](#_Toc484523979)

[7.2 Interaction diagrams 26](#_Toc484523980)

[1. Sequence Diagram 26](#_Toc484523981)

[Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass.  the diagrams are read left to right and descending 26](#_Toc484523982)

[2. Activity diagrams 26](#_Toc484523983)

[8 Implementation View 27](#_Toc484523984)

[9 Data View 28](#_Toc484523985)

[10 Deployment View 30](#_Toc484523986)

[11 Screen Design 31](#_Toc484523987)

[12 External Interface Architecture 32](#_Toc484523988)

[13 Glossary of Interfaces 33](#_Toc484523989)

**<Note: (This could go to design procedure)**

Use this (HLD) template for defining design considerations, High-level software architecture and high level views of the design.

Use DLD document for defining:

* Use case realization with Sequence / Collaboration diagrams, State diagrams and Class diagrams
* Package structure
* Components
* Class definition and specification of business logic for methods.
* Database ER Diagram, etc.

In case of small projects, complete design can be documented in HLD template itself.

While using tools like Rational Rose, Rational XDE follow one of the following options:

* Record HLD diagrams in HLD template, use tool generated document(s) as DLD / PS and provide reference to the documents in HLD template.
* Use DLD and PS templates and record tool generated diagrams / PS in the respective templates.

>

# Introduction & Brief Description

< Include here a brief description of the role and purpose of the HLD document. >

# References

<Include a description of related / referenced documents>

# Basic assumptions and limitations of the system

# System considerations, Architectural analysis, Design & Development decisions and rationale

<Include here a description of the architectural requirements, and design decisions from the perspective of requirements, development, testability and maintainability.

BASIC REQUIREMENTS

| **Criteria** | **Considerations / Alternatives** | **Design Decision and (optional) Rationale** |
| --- | --- | --- |
| GUI Design | Client Preferences, Existing Systems |  |
| Browser Compatibility (for Web applications) | IE, Netscape, Others |  |
| Quick Scalability | User, Location, Data Volume |  |
| Performance | Benchmark |  |
| H/w Platform Dependence / Independence | Intel / Sun / HP UX / SGI Itanium |  |
| OS Dependence / Independence | Windows / Solaris / Linux, etc |  |
| S/w Platform Dependence /Independence | App. Server, DB Server, Report Server etc. |  |
| Security | Role Based, SSL, PKI, Single sign-on |  |
| Client Supplied Components | Tested, Usable, Documented |  |
| 3rd Party Components (List all components) | Evaluation, Licence, etc.  (Describe consideration for each component) |  |
|  |  |  |

**ARCHITECTURE**

| **Architectural Elements** | **Considerations / Alternatives** | **Design Decision and (optional) Rationale** |
| --- | --- | --- |
| Architecture Type | Stand alone, Client Server, Web Based, MQ Based, Workflow etc. |  |
| Basis of Layering | No of tiers and purpose of each tier |  |
| Reusability | Use of existing components, Identify new components |  |
| Batch Operation | Manual, Scheduler based, Database related |  |
| Integration with External System(s) | For Input / Output, Process based, Message based |  |
| Load Balancing Mechanism (if applicable) | Clustering, Parallel servers |  |
| Fail over | Redundancy, Session Backup |  |
| Deployment Scenario | Location, Administration, Connectivity |  |
| Security | Firewall, Server organization |  |
| Troubleshooting | Event / Error logs,  Application instrumentation |  |
| Authentication | Standalone / LDAP integration |  |
| External Interface | SOA / Offline file based / EAI based |  |
|  |  |  |

**DESIGN**

| **Design Artifacts** | **Considerations / Alternatives** | **Details and (optional) Rationale** |
| --- | --- | --- |
| Personalization | Carry in Session or fetch when required? |  |
| Internationalization | Language, Currency |  |
| Time Zone | Transactional, Static Information |  |
| Transaction Handling | Programmatic, Transaction server (declarative) |  |
| Concurrency Control | Concurrent Transactions - Same user, Different users |  |
| User Message | Message Storage & Display mechanism |  |
| Paging Logic | Client Side, Server Side, Database hit |  |
| e-mail Notification | Auto-reply, send, receive, acknowledge |  |
| GUI | Applets, 3rd Party controls, Browser compatibility |  |
| Sorting / Searching | Client or server side,  Algorithm |  |
| Help | Static, Context Sensitive, User Manual |  |
| Error Handling  & Logging | Common Component, 3rd Party product |  |
| Messaging | Synch, Async,  Acknowledgement based |  |
| Security | Role Session, Encrypt, Account Lock, Usage Info |  |
| Cookies | Reason for using cookies |  |
| Validation | Client side, Server side |  |
| Audit Trail | Usage Level (when?), Format, Audit Report |  |
| Backup | Database, Files |  |
| Outage | Mechanism for safe handling and restoration |  |
| Network traffic | For high transaction modules |  |
|  |  |  |

**DATABASE DESIGN:**

|  |  |  |
| --- | --- | --- |
| **Database Aspects** | **Considerations / Alternatives** | **Details and (optional) Rationale** |
| Organization | Volume, Table Space, Table, View, Users |  |
| Database Side Processing | Use of SPs, Functions, Triggers, Cursors |  |
| Connection | Connection Pooling, Driver Type |  |
| Performance Tuning | Query Optimization, Indexing, De-Normalization |  |
|  |  |  |

**GUI / SCREEN DESIGN FOR WEB APPLICATIONS:**

| **GUI Aspects** | **Considerations / Alternatives** | **Details and (optional) Rationale** |
| --- | --- | --- |
| Usage of Frames | Yes / No |  |
| Usage of Style sheets (CSS) |  |  |
| Sitemap | Yes / No |  |
| Menu organization and Navigation | Side Menu / Top Menu / No Menu (Only URLs) |  |
| Technologies | ASP / JSP  Scripting Languages |  |
| Colour Scheme | Single scheme for all pages / Multiple schemes |  |
| Images and their size and quality | GIF / JPEG  Size |  |
| Max page size | Kbs |  |
| Usage of Page ID apart from Page title | Page ID Required / Not required |  |
| Screen Resolution | 800 x 600 / 1024 x 768 / both |  |
| Browser support | IE / Netscape |  |
| Validation | Common validation / Individual validation |  |
| Error message display locations |  |  |
| SSL certificate authority | Verisign / Webtrust |  |
| Support for disability | Screen reader support |  |
| Search Facility and Technology |  |  |
| Support for search engines |  |  |
| Usage of Data grids |  |  |
|  |  |  |

CODING:

| **Code Element** | **Considerations / Alternatives** | **Details and (optional) Rationale** |
| --- | --- | --- |
| Documentation | Javadoc, Manual |  |
| HTTP Method | GET, POST |  |
| Common Header & Footer and Includes |  |  |
| Session Cleaning | On logout, Window close |  |
| Database Connection | Close Open connections |  |
| Screen Validations | Date, Trim, Maxsize, Browser Compatibility |  |
| Security | Restricting intermediate URL access for valid & invalid users |  |
| Folder/ Package Structure |  |  |
|  |  |  |

**Testing:**

| **Testing aspects** | **Considerations / Alternatives** | **Details and (optional) Rationale** |
| --- | --- | --- |
| Testing Infrastructure | Design & Development considerations for testability |  |
| Testing framework (for automation) | Requirements for automation |  |
| Functional testing | Complex Business Logic |  |
| GUI Consistency | Browser compatibility, Resolution, UI Elements |  |
| Concurrency | Same user, Different User, Transactional/ Master Update |  |
| Stress (Concurrent Users) | Manual, Automated |  |
| Initial Configuration Test | Empty Database |  |
| Production Environment Simulation | Server Version, Patch Level, Load Balancing |  |
| Database Load Test |  |  |
|  |  |  |

**Maintainability:**

| **Maintenance aspects** | **Considerations / Alternatives** | **Details and (optional) Rationale** |
| --- | --- | --- |
| Super user | Required / Not required |  |
| Archival of past transaction data | Automatic / Manual |  |
| Error detection | Automatic log file / Admin enabled log files / Not required |  |
|  |  |  |

**Mapping of Critical performance parameters to design components:**

| **Sl No.** | **Performance Parameter** | **Requirement** | **Design Components meeting requirement** |
| --- | --- | --- | --- |
|  | Scalability | 1000 users and 200 concurrent user | Separate server hardware for Application, Reporting and Database connected by Gigabit Ethernet.  Windows 2003 server with IIS 6. Oracle Database. |
|  | Availability | 98% during Business Hours | Windows 2003 server with IIS 6. Oracle Database. Standby if unable to rectify within four hrs. |
|  | Portability | Nil |  |
|  | Extensibility | Integration with External system | Flat file based integration with batch update at midnight |

< This Introductory notes is provided as a general guideline and this can be removed from the document.

Architectural Representation

Introductory Notes:

General:

* Software architecture represents the structure of the software. This includes the structural arrangements of software components, and various static and dynamic interrelationships between these components.
* Software architecture is expressed using certain views, each of which serves a specific purpose. Each view is a specific abstraction of the architecture, for a specific purpose.
* Software architecture represents a high-level view of the system revealing the structure, but hiding all implementation details. Specifically, it reveals attributes such as responsibilities (of the constituents of the architecture), distribution, and deployment.
* Architecture should realize all the use case scenarios and should enable the stakeholders of the software to walk through the scenarios of each use case. This guarantees that the structure as represented by the architecture meets the functional requirements.
* It should present other systemic views to all the stakeholders of the software. Examples are - a component view for the development team, a network-centric deployment view for the network and hardware team, and a distribution-centric deployment view for the installation team etc.

The level of abstraction and number of views depends on the extent details the architect wishes to convey to the development team. UML and other diagrams can be used to convey the details with supporting textual description and rationale as applicable.

Recommended Architectural Views

**Logical or Conceptual View:**

* Provides conceptual organization of the software in terms of important classes and interfaces their organization in service packages and subsystems, and the organization of these subsystems into layers and packages. Also describe the frameworks.
* Standard and UML diagram can be used to depict architectural layers, package diagrams (with layers) and the interaction between layers, system sequence diagrams to show important use case realization scenarios
* Using this view, one should be able to walk through these abstractions to realize the use case scenarios.

Deployment View:

* This view depicts how various parts of the software are deployed (across various platforms) and how these parts communicate with each other. Physical deployment of processes and components to processing nodes, and the physical network configuration between nodes can also be part of deployment view.
* UML deployment diagrams can be used to depict deployment view.

Optional Architectural Views

Process View:

* This view captures the concurrency and synchronization aspects of the design. Shows processes and threads, their responsibilities, collaborations and the allocation of logical elements (layers, subsystems, classes …) to them.
* UML can be used to show class and interaction diagrams, state diagrams using UML process and thread notation.

Use case view:

* This view depicts the summary of architecturally significant use cases and their non-functional requirements, i.e. use cases that by their implementation illustrate significant architectural coverage or that exercise many architectural elements.
* UML use case diagram can be used to express architecturally significant use cases. Non-functional requirements are available in use case documents of the system requirements.

Data view:

* This view provides overview of persistent data schema, schema mapping from objects to persistent data (in relational DB), mechanism of mapping from objects to DB, Database stored procedures (SPs) and Triggers.
* UML class diagram can be used to describe a data model. ER diagrams also can be used to describe the data model

Implementation view:

* Implementation view is a summary description of noteworthy organization of deliverables and the things that create deliverables (such as source code). Implementation view model all code base including web pages, DLLs, executables, source code and so forth and their organization such as source code in java packages and bytecode organized in Jar files.
* UML package and component diagrams can be used to depict Implementation view .

Additional architectural views:

* Additional architectural views can be defined as required in a project. Eg. Technology architecture view, Security view, Interface architecture, etc. )

Software Architecture Goals

1. A design that is uniform and integrated.
2. A design that is parameterized and that reduces hard coding.
3. A design that results in elements, which can be reused within the application.
4. Breaking the application up into logical components that can be designed elegantly / easily
5. Dividing functionality among objects involved in maintaining and presenting data so as to minimize the degree of coupling between the objects without losing cohesion
6. Ensuring maintainability of the application

# Use-Case View

< Complete use case and diagrams are available in \_\_\_\_.doc. Non functional requirements are also described in the document. >

## Architecturally-Significant Use Case diagrams



# Logical View

< Provide brief description of Logical architecture >

<Describe the alternative architectures (and architectural components) evaluated and the rationale for selection of architecture adapted. Record the alternative solutions, decisions, influential factors and motivations for the noteworthy issues and decisions. Use DAR Procedure for major decisions (if any)>

## Application Architecture:

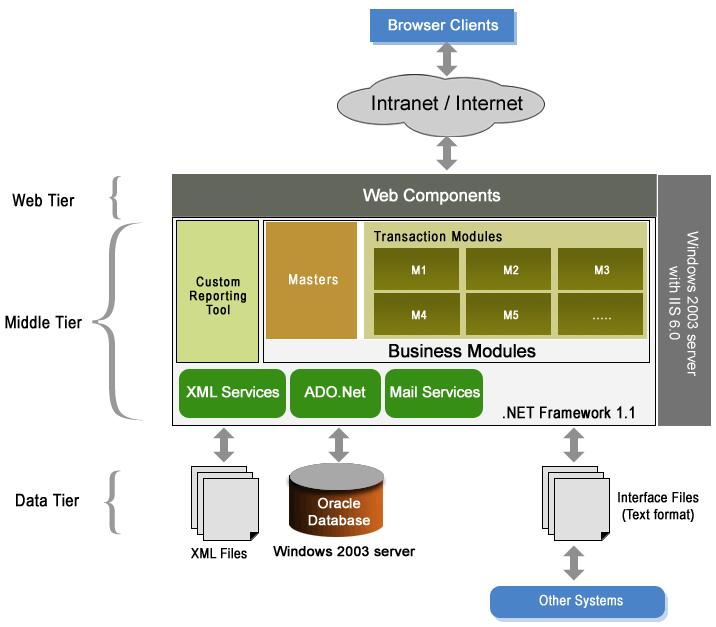
<Diagram below depicts the application architecture with subsystems

Multi-tiered approach

The application architecture is multi-tiered into:

1. Web tier
2. Middle tier and
3. Data tier >

Sample Diagram

****

## Architectural Pattern

We propose to use Model-View-controller (MVC) architecture pattern to achieve the architectural goals to the best possible extent.

MVC architecture adopted is a method for breaking an application up into logical components that can be architected easily. It is a way to divide functionality among objects involved in maintaining and presenting data so as to minimize the degree of coupling between the objects.

* **Model** represents the application data and the business rules that govern access and modification of this data. It notifies the view when it changes and provides the ability for the view to query the model about its status.
* **View** renders the contents of a model. It accesses data from the model and specifies how that data should be presented. When the model changes, it is the responsibility of the view to maintain consistency in its presentation.
* **Controller** defines application behavior. It interprets user gestures and maps them into actions to be performed by the model. Based on the user gesture and the outcome of the model, the controller selects a view to be rendered as part of the response to a user request.
* ASPX Pages
* Data Objects
* C#.Net classes
* Controller ASPX Page
* C#.Net classes

## Important layers

<Describe the layering concept and the description of each layer>

< The application architecture is further divided into six distinct logical layers as:

1. User Interface layer in web-tier
2. Controller Facade layer in Middle tier
3. Business components layer in Middle tier
4. Data Access object (DAO) layer in Middle tier
5. Model Update layer in Middle tier and
6. Application data and Stored procedures layer in Data tier

The above layers are explained below. They are elaborated in architecturally significant modal elements below.

**1. User Interface Layer**

This layer contains Application configuration file, Web Pages, XMLs (which contain the application information like screen flow etc.) and images. During application start, required application level data will be loaded from data tier and made available in application scope. Each request from the client is authenticated before invoking any of business operations.

**2. Controller Facade Layer**

This layer consists of Web controller and Facade controller. Web controller acts as a single point contact, which receives every page request from browser and passes on the control to Facade controller. Facade controller takes over the control from Web controller and delegates the controls to the appropriate classes.

For user interface layer, Controller Façade is a single **entry / exit** point for the business rules in the application. This is responsible for handling page specific data from the session and reloading the page with the required data that is fetched by the DAO.

**3. Business Components Layer**

These components contain the business logic of application features. These contact Data Access Component to retrieve data for a request. Depending on business validations output is returned to Facade layer.

**4. Data Access Object (DAO) Layer**

This component provides data services to the Business component. A common Class takes query string as an argument and returns a Dataset.

**5. Model Update Layer**

This would organize the required data for the next screen.

**6. Data and Stored procedures Layer**

This layer contains the application data and Stored procedures that manipulate data at data server end.

>

## Architecturally-Significant Model Elements

Following diagram shows the architecturally significant elements in the architectural framework.



Façade Controller

BO Handler

HLD\_OOAD

Document Version / Details : Ver. 2.22/ 15-Jun-2017

WebController

The details of the above elements are given below.

**WfrmControlForm.aspx**

Webcontrolform.aspx is to control the web form flow. This will decide the next screen to be shown in browser. The screen flow details would be maintained in an XML file. Webcontrolform will transfer the control to Façade controller. If the current operation does not require any business logic/data access, for e.g. to show a screen, which doesn’t require any data, that page will be shown to the user by Webcontrolform.aspx, bypassing Facade controller.

**cFaçade Controller**

Façade controller is the main controller for the system. This controller will transfer the control to different layers for business logic/data processing.

**cBOEntry**

BOEntry loads the required BOHandler class for processing.

**cBOHandler**

All business logic will get executed here in this layer. This class contains the business logic for the selected operation. If Data access is required for this operation, control will be transferred to DAO Layer.

**cDAOEntry**

DAOEntry loads the required DAOHandler class for Processing.

**cDAOHandler**

All queries related to database (SQL statements or stored procedures) will be managed in this class. This makes use of Data Access Component for database operations.

**cDataAccess**

Usage of Data Access Component is to connect to database and execute various queries. Data Access Component (DAC) is a managed component that exposes different methods to execute different types of queries. It uses the ADO.NET for data access.

**cModelUpdate**

This class will load the required ModelUpdatehandler class for processing.

**cModelUpdateHandler**

This would organize the required data for the next screen.

## Package and Subsystem Layering

<Describe the package and subsystem layering>

Following diagram depicts the modular / package organization of the application.



## Architectural Description

<

The framework of classes that have are designed to adopt MVC design pattern would have the following characteristics.

* Each user action decides the flow in the architecture
* User interface transfers the controller to the next level of framework
* Framework classes of Façade, BO and DAO have an entry point and delegation. This delegation is achieved through XML mappings.
* Data value object is used for transmittal of data across layers. This will help to handle data effectively across layers.
* Data retrieved by DAO is returned calling classes in terms of hash tables.
* Model update will clear previous page contents from the session and update with the retrieved one.
* If action performed is successful then Control Form loads new screen or else the user is redirected to an error page.

Constraints

* Compatibility of .NET framework 1.1 with browser software & its versions and other software that are used in the system.
* Compatibility of reporting tool for generating graphs and charts with .NET framework 1.1. Ability of reporting tool to generate the types of charts that is required by the application.

>

# Process View

## Class diagram

The following Class diagram describes the process view of the application architecture.



The details of the above elements are given below.

*(It is to be noted that few of the elements that have been discussed above have been reproduced once again in this section also, for readability)*

**xxxx.aspx**

This is the user screen for data entry. After entering data, user will submit this page and the control will go to its code behind file. (xxxx.aspx.cs).

**xxxx.aspx.cs**

This page will create a DVOxxxx object. This page is inherited from webControlForm.

**cDVOxxxx.cs**

This object contains the form input data.

**wfrmControlForm.aspx**

Webcontrolform.aspx is to control the web form flow. This will decide the next screen to be shown in browser. The screen flow details would be maintained in an XML file. Webcontrolform will transfer the control to Façade controller. If the current operation does not require any business logic/data access, for e.g. to show a screen, which doesn’t require any data, that page will be shown to the user by Webcontrolform.aspx, bypassing Facade controller.

**wfrmControlForm.aspx.cs**

(This is the class that acts as “Code-behind” for wfrmControlForm.aspx).

This transfers the control to Façade controller along with action id and DVOxxxx object. Once the Façade processing is over the control form will load the next resulting page for the user.

**cFaçadeController**

Façade controller is the main controller for the system. This controller will transfer the control to different layers for business logic/data processing.

This controller transfers the control to BOEntry/ DAOEntry based on the logic along with action ID and DVOxxxx. Once the BO/DAO processing is over it will transfer the control to Model Update along with action ID and Hash Table object, which contains the resulting data.

**cBOEntry**

BOEntry loads the respective BOHandler class(BOxxxx) for processing.

**cBOHandler**

All business logic will get executed here in this layer. This class contains the business logic for the selected operation. If Data access is required for this operation, control will be transferred to DAO Layer.

This is the base class for all BOxxxx classes.

**cBOxxxx**

This class contains the business logic for the specified action. The form data required for Business processing will be extracted from the DVOxxxx object. Then the control will be transferred to DAOEntry for database operations along with actionid and DVOxxxx object.

**cDAOEntry**

DAOEntry class loads the respective DAOHandler class(DAOxxxx) for processing.

**cDAOHandler**

All queries related to database (SQL statements or stored procedures) will be managed in this class. This makes use of Data Access Component for database operations.

This is the base class for all DAOxxxx classes.

**cDAOxxxx**

This class contains the data access logic for the specified action. The form data required for this data processing will be extracted from DVOxxxx object. All queries related to database operations are done here.

**cDataAccess**

Usage of Data Access Component is to connect to database and execute various queries. Data Access Component (DAC) is a managed component that exposes different methods to execute different types of queries. It uses the ADO.NET for data access.

This class establishes database connection and does database operations and return the resultant data to the previous layer.

**cModelUpdate**

This class will load the required ModelUpdatehandler class (ModelUpdatexxxx) for processing.

**cModelUpdateHandler**

This class is the base class for all ModelUpdatexxxx classes.

**cModelUpdatexxxx**

This would organize the required data for the next screen. This class will retrieve the data from the hashtable, which was received from façade controller. Then it will create a new hashtable and upload that hash table into session and in turn returns to the Façade controller.

## Interaction diagrams

Interaction diagrams model the behavior of  use cases by describing the way groups of objects interact to complete the task.

The two kinds of interaction diagrams are sequence and collaboration diagrams

## Sequence Diagram

## Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass.  the diagrams are read left to right and descending

Collaboration diagram

Collaboration Diagram show the relationship between objects and the order of messages passed between them.

## Activity diagrams

Activity diagrams should be used in conjunction with other modeling techniques such as [interaction diagrams](http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/interaction.htm) and [state diagrams](http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/state.htm).  The main reason to use activity diagrams is to model the workflow behind the system being designed

# Implementation View

< The following Component diagram describes the Implementation view of the application. This diagram indicates the various layers in componentized way.

(Example 1)



(Example 2)



# Data View

Application will use Oracle 9i database to store the project data. Oracle 9i database reliable, maintainable and flexible with high performance.

Database Design approach

The approach for the DB design is as follows:

1. The ERWin tool is used for Data modeling and prepare ER diagram. This tool is useful for effective design, to implement and maintain high quality databases, to visualize and efficiently define the data requirements to ensure alignment with Janus requirements. Using this tool, the quality and maintainability can be improved significantly.
2. Stored procedures would be designed during the DLD phase.

### Proposed Architecture

Business logic components

Stored Procedures

Oracle 9i Database

Database architecture diagram

The complex database operations will be handled by stored procedures. Hence the business logic components call the procedure with the required inputs, and the stored procedure takes care of the data operations (Please refer the above diagram). Simple operations will be done with direct interaction with the database. The stored procedures will be detailed and finalized at the detailed level design stage

Stored procedures help in:

1. Improving Performance by faster processing of large volume of data
2. Reducing network traffic and latency, thus improving response time and availability.

### Database model

The following documents describe the database design model and design details.

1. DBModel describes the database design model
2. Table definitions describes the database design details

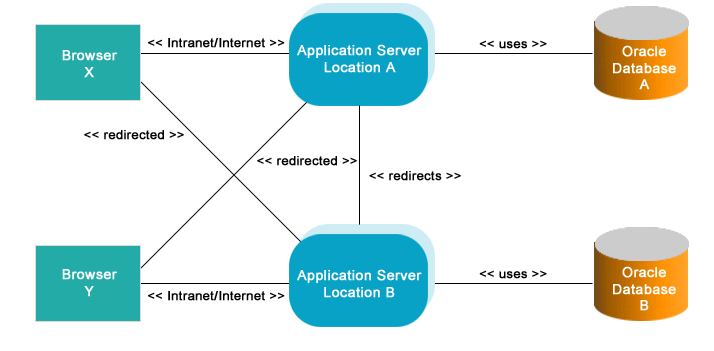
# Deployment View

<

Deployment architecture describes the system in terms of the allocation of processes to processing units and network configuration.

>

(Example 1)



(Example 2)

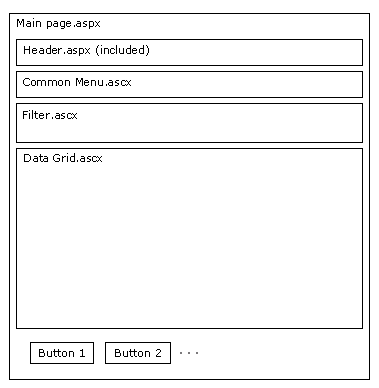


# Screen Design

<Define the typical screen design layout, style guide to be followed, CSS to be used, etc. in this section >

< Each screen in the application is an aspx file. The contents of a screen may vary from user to user. If a screen has to display totally a different content for one user than the other, two different aspx files will be created. If there is a very little change in the contents like a button need not be shown for a particular role, there will be only one aspx file, which will be rendered differently for users of different role. Each screen comprises of the following components

1. Header (appears in all screen)
2. Menu (appears in all screen)
3. Filter options (only in specific screens)
4. Data Grid (only in specific screens)
5. Action buttons (almost in all screens)

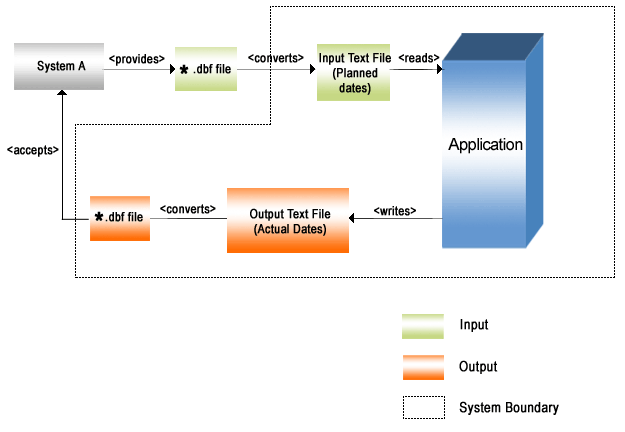


>

# External Interface Architecture

< Describe the architecture for interface with External system. Describe the assumptions & limitations specific to external interface>

(Example )



# Glossary of Interfaces

<List of Interfaces created during HLD.>

|  |  |  |  |
| --- | --- | --- | --- |
| SL No | Class Name | Interface Name | Remarks |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |